Propositional and Predicate Logics of Incomplete Information

Propositional and Predicate Logics of Incomplete Information

Marco Console and **Paolo Guagliardo** and **Leonid Libkin**
School of Informatics, University of Edinburgh

Abstract

One of the most common scenarios of handling incomplete information occurs in relational databases. They describe incomplete knowledge with three truth values, using Kleene's logic for propositional formulae and a rather peculiar extension to predicate calculus. This design by a committee from several decades ago is now part of the standard adopted by vendors of database management systems. But is it really the right way to handle incompleteness in propositional and predicate logics?

Our goal is to answer this question. Using an epistemic approach, we first characterize possible levels of partial knowledge about propositions, which leads to six truth values. We impose rationality conditions on the semantics of the connectives of the propositional logic, and prove that Kleene's logic is the maximal sublogic to which the standard optimization rules apply, thereby justifying this design choice. For extensions to predicate logic, however, we show that the additional truth values are not necessary: every many-valued extension of first-order logic over databases with incomplete information represented by null values is no more powerful than the usual two-valued logic with the standard Boolean interpretation of the connectives. We use this observation to analyze the logic underlying SQL query evaluation, and conclude that the many-valued extension for handling incompleteness does not add any expressiveness to it.

Introduction

Incomplete information is ubiquitous in applications that involve querying and reasoning about data. It is one of the oldest topics in database research (Codd 1975), and is essential in many applications such as data integration (Lenzerini 2002), data exchange (Arenas et al. 2014), inconsistent databases (Bertossi 2011), and ontology-based data access (Bienvenu and Ortiz 2015).

When it comes to querying incomplete data, practical solutions, such as relational databases, rely on *many-valued logics* to properly account for the lack of certainty. In fact, every database management system (DBMS) uses a three-valued logic for handling incomplete information, namely Kleene's logic (Bolc and Borowik 1992). This was the design choice of SQL, the language of relational DBMSs, which is now written into the SQL Standard (ISO/IEC 2016), presented in all database textbooks, and implemented in all database products. However, this is far from the only logic to have been considered for representing incomplete information. The use of Kleene's logic was first proposed by Codd (1975), but many other variants appeared afterward. Codd (1987) looked at a four-valued logic, but in the end argued against it due to the additional complexity. Nonetheless, well-documented problems with incomplete information (Date and Darwen 1996; Date 2005) led to the search of more appropriate logics for handling incompleteness. For example, Gessert (1990) revisited four-valued logics, while Yue (1991) considered logics with four, five, and seven values, and showed how to encode them with three. A different kind of four-valued logics for missing data was studied by Console, Guagliardo, and Libkin (2016), while Darwen and Date (1995) suggested dropping nulls altogether and go back to the usual Boolean two-valued logic.

There is also no shortage of many-valued logics that have been proposed in closely related contexts. For example, a variety of many-valued logics were used in the study of default reasoning (Reiter 1980) or in reasoning about inconsistency (Zamansky and Avron 2006). Those are typically based on the notion of bilattices, providing truth and knowledge orderings on the truth values (Arieli and Avron 1996; Ginsberg 1988). A common one is Belnap's bilattice with four truth values (Belnap 1977; Arieli and Avron 1998), which also found database applications (Grahne, Moallemi, and Onet 2015); but others exist as well, e.g., many generalizations of Kleene's logic based on numerical intervals describing the degree of being true (Fitting 1991). A many-valued propositional logic must also provide an interpretation of propositional connectives. To make the general picture even muddier, for different sets of truth values, different semantics of propositional connectives exist, sometimes even non-deterministic ones (Arieli, Avron, and Zamansky 2010).

Thus, we are far from having a clear picture of what to use as a logic of incomplete information in data management applications. Choices are numerous, and there is no final argument as to why the approach of DBMSs that use Kleene's logic is the right one. Hence, the first question we address is:

1) *What is the right many-valued propositional logic for handling incomplete information?*

Now suppose we have a propositional logic that correctly accounts for truth values of statements about incomplete information, and for operations on them. In querying data, however, we use *predicate* logics. Indeed, the core of SQL is essentially a programming syntax for relational calculus, which is another name for first-order (FO) predicate logic.

Of course we know how to lift the semantics of propositional logic to the full predicate calculus by treating existential and universal quantifiers as disjunctions and conjunctions over all elements of the universe. What we do not know is how different choices of propositional logic for incomplete information affect the power of predicate calculus. As one example, consider the version of FO that underlies SQL and is based on Kleene's logic. What extra power does it possess over FO under the usual two-valued Boolean interpretation of the connectives? It was recently argued, by means of rewriting SQL queries, that FO based on Kleene's logic can be encoded in the usual Boolean FO (Guagliardo and Libkin 2017). But is there a general result in logic that underlies such a translation, and what is so special about Kleene's logic that makes it work?

Even more generally, the second question we would like to address is:

2) *How does the choice of a propositional logic for incomplete information affect predicate logic?*

Finally, we would like to understand how these theoretical considerations relate to the practice of incomplete data in relational databases. A rough approximation of the core of SQL – the way it is presented in many database textbooks – is first-order logic. But as soon as incomplete information enters the picture, this becomes a many-valued FO. And yet there is even more to it: in SQL queries, answer tuples are split into *true* ones that need to be returned, and others that are not returned, thus collapsing a three-valued logic to two-valued. This leads to our last question:

3) *What is the logic that underlies real-life handling of incomplete information in relational databases (i.e., SQL's logic), and how much more power than the usual two-valued* FO *does it possess?*

The goal of this paper is to address these three questions. Below we outline our main contributions.

Propositional logic To understand what a proper propositional logic for reasoning about incomplete information is, we need to define its truth values, and truth tables for its connectives (we shall concentrate on the standard ones, i.e., $\wedge$, $\vee$, and $\neg$, although we shall see others as well). We follow the approach of Ginsberg (1988) to turn partial knowledge about the truth of a proposition into truth values. If we have a set W of worlds, and two of its subsets T and F in which a proposition is true and false, respectively, this produces a description (T, F, W). It is possible that $T \cup F \neq W$, i.e., we may have partial knowledge about the truth or falsity of a proposition. We require however that $T \cap F = \varnothing$, as here we do not consider inconsistent descriptions.

Taking those descriptions (T, F, W) directly as truth values, however, is not satisfactory: we shall have too many of them. Instead, we want to take as truth values *what we know* about such descriptions.

We abstract this knowledge as *epistemic theories* of such descriptions: they say what is known about a proposition being possibly or certainly true or false. Then, as truth values we take maximally consistent epistemic theories. We show that there are only six such theories, resulting in a six-valued logic $\mathbb{L}_{6v}$. Its truth tables are again very naturally derived from epistemic theories of partial knowledge about truth of propositions.

As a final step, we then look at what makes a many-valued logic database friendly. It needs to be a sublogic of $\mathbb{L}_{6v}$ and yet satisfy some basic equivalences we expect to hold to be able to perform query evaluation and optimization. We then show that the maximal sublogic of $\mathbb{L}_{6v}$ that satisfies those equivalences is $\mathbb{L}_{3v}$, the three-valued logic of Kleene used in all commercial DBMSs. Thus, we justify the choice that was made by SQL designers and standards committees in choosing $\mathbb{L}_{3v}$ as the logic to be implemented in all database products.

Predicate logic We have justified Kleene's logic $\mathbb{L}_{3v}$ as the right choice for handling incompleteness in database contexts. But database languages are not propositional: they are based on FO instead. Thus, we next look at variants of FO based on propositional many-valued logics such as $\mathbb{L}_{3v}$ and $\mathbb{L}_{6v}$, and compare their power with that of the usual Boolean FO (denoted by BFO from now on), based on just two values **t** and **f**. Our main result is that when added to FO, these many-valued propositional logics add no power: FO based on $\mathbb{L}_{3v}$, or on $\mathbb{L}_{6v}$, or on any other many-valued logic (under some mild restrictions on the connectives) has no more power than BFO.

The logic of SQL We finally apply the above observation to SQL's logic. We explain that it corresponds to a $\mathbb{L}_{3v}$-based FO with an extra connectives that allows one to collapse truth values **f** and **u** into one, but it still has no more power than BFO. Thus, even though SQL designers were justified in choosing Kleene's logic as the propositional logic for reasoning about incomplete information, they overlooked the fact that, when considered within FO, such a logic does not add any expressive power.

To sum up, our investigation validates the choice of Kleene's logic by the designers of SQL, but at the same time asks whether it was really necessary and opens up a possibility for future languages that handle incomplete information to avoid the recourse to many-valued logics. Notice that much of the criticism of SQL concentrated on its propositional logic. However we showed that it was very reasonable: a six-valued logic would have been better justified but the three-valued logic is better at handling computational aspects. For predicate logics, our results say that these many-valued logics could have been avoided altogether. However, the price for this is a different way of expressing logical queries, and thus this result is of more interest for future language design rather than changing the current choices.

Organization The paper is structured around three main themes: propositional logics, predicate logics, and the logic of SQL, followed by conclusions and future work.

Propositional Logics

Our study of logics for incomplete information starts at the propositional level. The goal of this section is to define a propositional logic for handling incompleteness, with a special regard to applications that deal with incomplete data, including relational databases query languages.

To this end, we first need to formally define propositional formulae. We assume a countably infinite set of symbols, referred to as *propositional atoms*. For a set Ω of connectives with associated (positive) arities, the *propositional language* $\mathcal{L}$ over Ω is defined inductively as follows: every propositional atom is a formula of $\mathcal{L}$; if ω is an n-ary connective in Ω and $\alpha_1, \ldots, \alpha_n$ are formulae of $\mathcal{L}$, then so is $\omega(\alpha_1, \ldots, \alpha_n)$; nothing else is in $\mathcal{L}$. We assume that the binary connectives $\wedge$ and $\vee$, for which we use the infix notation, and the unary connective $\neg$ are always present. As will be relevant in the next section, this general definition allows for the inclusion of additional connectives in the language.

The standard way of evaluating propositional formulae is to associate atoms with *truth values*, which are then propagated through the connectives by means of *truth tables*. We define a *(propositional) logic* $\mathbb{L}$ as a pair $(\mathbf{T}, \Omega)$, where $\mathbf{T}$ is the set of truth values and Ω is the set of truth tables, which are functions $\omega\colon \mathbf{T}^n \to \mathbf{T}$, of appropriate arities, associated with the connectives. We say that $\mathbb{L}$ is a logic for a language $\mathcal{L}$ if $\mathbb{L}$ defines truth tables for every connective of $\mathcal{L}$. With a deliberate abuse of notation, we denoted by Ω both the connectives of $\mathcal{L}$ and the truth tables associated with them in $\mathbb{L}$. When it is not clear from the context, we use $\omega^{\mathbb{L}}$ to explicitly denote the truth table of $\mathbb{L}$ for the connective ω.

Given a logic $\mathbb{L} = (\mathbf{T}, \Omega)$ for a language $\mathcal{L}$, and a mapping μ from propositional atoms to the truth values in $\mathbf{T}$, the evaluation of a formula $\alpha \in \mathcal{L}$ under μ in $\mathbb{L}$ is the truth value $\mathsf{tv}_{\mathbb{L}}(\alpha, \mu)$ in $\mathbf{T}$ defined inductively as follows:

$$\mathsf{tv}_{\mathbb{L}}(\alpha, \mu) = \mu(\alpha) \text{ if } \alpha \text{ is a propositional atom,}$$
$$\mathsf{tv}_{\mathbb{L}}\big(\omega(\alpha_1, \ldots, \alpha_n), \mu\big) = \omega^{\mathbb{L}}\big(\mathsf{tv}_{\mathbb{L}}(\alpha_1, \mu), \ldots, \mathsf{tv}_{\mathbb{L}}(\alpha_n, \mu)\big),$$

for every $\alpha, \alpha_1, \ldots, \alpha_n \in \mathcal{L}$ and every n-ary connective ω.

For $\Omega = \{\wedge, \vee, \neg\}$, the standard Boolean logic $\mathbb{L}_{\text{Bool}}$ has truth values $\{\mathbf{t}, \mathbf{f}\}$ and truth tables as in Figure 1, while SQL uses Kleene's three-valued logic, denoted by $\mathbb{L}_{3\text{v}}$, with truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ and truth tables as in Figure 2. But is $\mathbb{L}_{3\text{v}}$ the right propositional logic to deal with incomplete information in relational databases? To answer this question, we first need an appropriate model of incompleteness; then, we must define what kind of information truth values represent in this model, and how many of them are needed; finally, we need to define truth tables for $\wedge$, $\vee$ and $\neg$ that propagate information in a consistent way.

Model of Incompleteness

In many data management applications, especially those involving knowledge representation and reasoning, the veracity of data is a common problem. This results in dealing with

$\wedge$	**t**	**f**
t	**t**	**f**
f	**f**	**f**

$\vee$	**t**	**f**
t	**t**	**t**
f	**t**	**f**

	$\neg$
t	**f**
f	**t**

Figure 1: The truth tables of $\mathbb{L}_{\text{Bool}}$.

$\wedge$	**t**	**f**	**u**
t	**t**	**f**	**u**
f	**f**	**f**	**f**
u	**u**	**f**	**u**

$\vee$	**t**	**f**	**u**
t	**t**	**t**	**t**
f	**t**	**f**	**u**
u	**t**	**u**	**u**

	$\neg$
t	**f**
f	**t**
u	**u**

Figure 2: The truth tables of $\mathbb{L}_{3\text{v}}$.

two main sources of incomplete information: first, queries must be answered over sets of possible worlds and, second, the answer to a query may not be well defined, or known, in some of them. In the literature on many-valued logics, the approach of (Ginsberg 1988) accounts for both these sources of incomplete information, and we follow it here as a basis for our model. However, as we shall discuss later on in this section, we deviate from Ginsberg's approach with respect to what truth values are and represent.

A *propositional interpretation* $\mathcal{I}$ is a triple (t, f, W), where W is a non-empty set of *worlds*, and t and f are functions from $\mathcal{L}$ to the powerset of W such that, for every $\alpha, \beta \in \mathcal{L}$, all of the following hold:

$$t(\alpha) \cap f(\alpha) = \varnothing \;; \qquad t(\alpha \wedge \beta) = t(\alpha) \cap t(\beta) \;;$$
$$f(\neg\alpha) = t(\alpha) \;; \qquad t(\alpha \vee \beta) = t(\alpha) \cup t(\beta) \;;$$
$$t(\neg\alpha) = f(\alpha) \;; \qquad f(\alpha \wedge \beta) = f(\alpha) \cup f(\beta) \;;$$
$$f(\alpha \vee \beta) = f(\alpha) \cap f(\beta) \;.$$

Intuitively, t tells us on which worlds a given formula is true, while f indicates where it is false. When a world w is neither in $f(\alpha)$ nor in $t(\alpha)$, the formula α is said to be *undefined* in w.

In (Ginsberg 1988), objects similar to propositional interpretations defined above are used as truth values for formulae. This is incompatible with the standard evaluation of formulae we defined earlier. Instead, we want to collate the information provided by propositional interpretations and abstract it as truth values. In a sense, a truth value should represent what we know about formulae with respect to interpretations. To formalize this intuition, it is natural to make use of some form of epistemic logic.

Given a propositional language $\mathcal{L}$, the language $\mathcal{L}^{\mathbf{KP}}$ of epistemic formulae is defined inductively as follows:

- every propositional formula in $\mathcal{L}$ is in $\mathcal{L}^{\mathbf{KP}}$;
- if φ and ψ are in $\mathcal{L}^{\mathbf{KP}}$, then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\neg\varphi$;
- if φ is in $\mathcal{L}^{\mathbf{KP}}$, then so are $\mathbf{K}\varphi$ and $\mathbf{P}\varphi$;
- nothing else is in $\mathcal{L}^{\mathbf{KP}}$.

An epistemic formula is said to be *subjective* if every propositional atom appears in the scope of $\mathbf{K}$ or $\mathbf{P}$.

The semantics of epistemic formulae is given with respect to a propositional interpretation $\mathcal{I} = (t, f, W)$ and an *appointed* world $w \in W$. Whether $\mathcal{I}$ and w satisfy $\varphi \in \mathcal{L}^{\mathbf{KP}}$, written $\mathcal{I}, w \models \varphi$, is inductively defined as follows:

- $\mathcal{I}, w \models \alpha$ if $w \in t(\alpha)$, for $\alpha \in \mathcal{L}$;
- $\mathcal{I}, w \models \mathbf{K}\varphi$ if $\mathcal{I}, w' \models \varphi$ for every $w' \in W$;
- $\mathcal{I}, w \models \mathbf{P}\varphi$ if $\mathcal{I}, w' \models \varphi$ for some $w' \in W$;
- $\mathcal{I}, w \models \neg\varphi$ if $\mathcal{I}, w \not\models \varphi$;
- $\mathcal{I}, w \models \varphi \wedge \psi$ if $\mathcal{I}, w \models \varphi$ and $\mathcal{I}, w \models \psi$;
- $\mathcal{I}, w \models \varphi \vee \psi$ if $\mathcal{I}, w \models \varphi$ or $\mathcal{I}, w \models \psi$.

Observe that whether a subjective formula φ is satisfied in a propositional interpretation $\mathcal{I}$ does not depend on the choice of the appointed world, hence we simply say that $\mathcal{I}$ satisfies φ, and write $\mathcal{I} \models \varphi$. Moreover, we denote by $\mathsf{Mod}(\varphi)$ the set of all models of φ, i.e., propositional interpretations that satisfy φ. We say that φ is satisfiable whenever $\mathsf{Mod}(\varphi)$ is non-empty.

We also remark that, unlike the standard operators $\Box$ and $\Diamond$ in classical modal logic, $\mathbf{K}$ and $\mathbf{P}$ here are not dual: while $\mathbf{K}\varphi$ implies $\neg\mathbf{P}\neg\varphi$, the converse is not necessarily true. To see this, consider a propositional formula α and the interpretation $\mathcal{I} = (t, f, \{w_1, w_2\})$ such that $t(\alpha) = \{w_1\}$ and $f(\alpha) = \varnothing$; then, it is easy to verify that $\mathcal{I}$ satisfies $\neg\mathbf{P}\neg\alpha$ but not $\mathbf{K}\alpha$, because $w_2 \notin t(\alpha)$.

Truth Values

We need to understand what it means for a propositional formula to be true or false in a propositional interpretation. To do that, we resort to the notion of modalities.

Given a propositional formula α, the modalities of α are the modal formulae $\mathbf{K}\alpha$, $\mathbf{P}\alpha$, and their negation. Intuitively, the modalities of α describe the way α is true on a given propositional interpretation. To define truth values, then, we will look at the modalities of propositional formulae and their negations.

More formally, for a propositional formula α, we denote by $\mathcal{M}(\alpha)$ the set consisting of all modalities of α and $\neg\alpha$. A subset M of $\mathcal{M}(\alpha)$ is called *consistent* if there exists at least one propositional interpretation $\mathcal{I}$ for which every formula in M is satisfied. A subset of $\mathcal{M}(\alpha)$ is *maximally consistent* if, in addition, none of its proper supersets is a consistent subset of $\mathcal{M}(\alpha)$.

Intuitively, every maximally consistent subset of $\mathcal{M}(\alpha)$ defines a possible way in which a propositional formula can be evaluated on a propositional interpretation. Thus, to capture all possibilities, we need as many truth values as there are maximally consistent subsets of $\mathcal{M}(\alpha)$. The following shows that our propositional logic must be six-valued.

Theorem 1. *For every propositional formula α, there are at most 6 maximally consistent subsets of $\mathcal{M}(\alpha)$. These are:*

$$\{\ \mathbf{K}\alpha,\ \mathbf{P}\alpha, \neg\mathbf{K}\neg\alpha, \neg\mathbf{P}\neg\alpha\ \} \quad (1)$$

$$\{\neg\mathbf{K}\alpha, \neg\mathbf{P}\alpha,\ \mathbf{K}\neg\alpha,\ \mathbf{P}\neg\alpha\ \} \quad (2)$$

$$\{\neg\mathbf{K}\alpha,\ \mathbf{P}\alpha, \neg\mathbf{K}\neg\alpha,\ \mathbf{P}\neg\alpha\ \} \quad (3)$$

$$\{\neg\mathbf{K}\alpha,\ \mathbf{P}\alpha, \neg\mathbf{K}\neg\alpha, \neg\mathbf{P}\neg\alpha\ \} \quad (4)$$

$$\{\neg\mathbf{K}\alpha, \neg\mathbf{P}\alpha, \neg\mathbf{K}\neg\alpha,\ \mathbf{P}\neg\alpha\ \} \quad (5)$$

$$\{\neg\mathbf{K}\alpha, \neg\mathbf{P}\alpha, \neg\mathbf{K}\neg\alpha, \neg\mathbf{P}\neg\alpha\ \} \quad (6)$$

Proof. Let $\mathcal{I} = (t, f, W)$ be a propositional interpretation. If $\mathcal{I}$ satisfies $\mathbf{K}\alpha$, then by the assumption that $W \neq \varnothing$ it also satisfies $\mathbf{P}\alpha$, $\neg\mathbf{K}\neg\alpha$ and $\neg\mathbf{P}\neg\alpha$. Thus, we get (1).

Otherwise, when $\mathcal{I} \not\models \mathbf{K}\alpha$, $\mathcal{I}$ may or may not satisfy $\mathbf{P}\alpha$. If it does, then $\mathcal{I} \not\models \mathbf{K}\neg\alpha$. Under this assumption, we have two possibilities: either $\mathcal{I}$ satisfies $\mathbf{P}\neg\alpha$, in which case we get the set (3), or not, and we get (4).

Suppose now $\mathcal{I} \not\models \mathbf{K}\alpha$ and $\mathcal{I} \not\models \mathbf{P}\alpha$. If $\mathcal{I}$ satisfies $\mathbf{K}\neg\alpha$, then by the assumption that $W \neq \varnothing$ it also satisfies $\mathbf{P}\neg\alpha$. Thus, we get the set (2).

Finally, if $\mathcal{I} \not\models \mathbf{K}\neg\alpha$, then $\mathcal{I}$ may or may not satisfy $\mathbf{P}\neg\alpha$. Thus, we get the sets (5) and (6), respectively. $\square$

We now analyze the information each of the above sets gives us for an arbitrary propositional formula α, and abstract it as a truth value, referring to the six maximally consistent sets in Theorem 1.

(1) We know that α is *true in all worlds* ($\mathbf{K}\alpha$). We abstract this as the truth value **t** (*always true*).

(2) We know that $\neg\alpha$ is true in all worlds ($\mathbf{K}\neg\alpha$), hence α is *false in all worlds*. We abstract this as the truth value **f** (*always false*).

(3) We know that there exists a world w in which α is true ($\mathbf{P}\alpha$) and there exists a world w' in which its negation is true ($\mathbf{P}\neg\alpha$). Since α cannot be both true and false in the same world, we have $w \neq w'$. We abstract this as the truth value **s** (*sometimes true and sometimes false*).

(4) We know that there is a world in which α is true ($\mathbf{P}\alpha$) but we do not know whether there is a (distinct) world in which its negation is true ($\neg\mathbf{P}\neg\alpha$). Thus, α could be true in all worlds, but we do not know that ($\neg\mathbf{K}\alpha$). We abstract this as the truth value **st** (*sometimes true*).

(5) We know that there is a world in which the negation of α is true ($\mathbf{P}\neg\alpha$) and where α is then false, but we do not know whether there is a (distinct) world in which α is true ($\neg\mathbf{P}\alpha$). Thus, α could be false in all worlds, but we do not know that ($\neg\mathbf{K}\neg\alpha$). We abstract this as the truth value **sf** (*sometimes false*).

(6) We do not know whether there exists a world in which α is true ($\neg\mathbf{P}\alpha$) nor whether there is one where its negation is true ($\neg\mathbf{P}\neg\alpha$). That is, we have no information at all, and we abstract this as the truth value **u** (*unknown*).

Thus, our set of truth values is $\mathbf{T}_{6\mathrm{v}} = \{\mathbf{t}, \mathbf{f}, \mathbf{s}, \mathbf{st}, \mathbf{sf}, \mathbf{u}\}$.

With each truth value τ and each propositional formula α, we associate the (subjective) epistemic formula χ^{τ}_{α} given by the conjunction of all formulae in the maximally consistent subset of $\mathcal{M}(\alpha)$ corresponding to τ. So, for example, $\chi^{\mathbf{s}}_{\alpha}$ is the conjunction of all formulae in (3), that is, $\neg\mathbf{K}\alpha \wedge \mathbf{P}\alpha \wedge \neg\mathbf{K}\neg\alpha \wedge \mathbf{P}\neg\alpha$. Intuitively, the satisfiability of χ^{τ}_{α} tells us whether it is possible for α to evaluate to the truth value τ.

Truth Tables

With the set of truth values in place, we now look at how the truth tables for the connectives are defined. Starting from the fact that truth values correspond to maximally consistent sets of modalities, we will argue that the truth tables must satisfy two reasonable requirements: *consistency* and *generality*.

Consistency Let us first consider the unary connective $\neg$; given a truth value τ, which truth value should $\neg\tau$ denote? If τ is **t**, intuition tells us that $\neg\tau$ should not be **t**. Indeed, such a situation cannot occur, in the sense that for every propositional formula α there exists no interpretation $\mathcal{I}$ that satisfies both $\chi^{\mathbf{t}}_{\alpha}$ and $\chi^{\mathbf{t}}_{\neg\alpha}$.

For binary connectives, the situation is similar; for example, $\mathbf{t}\wedge\mathbf{t}$ should not be **f**, as it cannot happen that for propositional formulae α and β there exists an interpretation $\mathcal{I}$ that satisfies $\chi^{\mathbf{t}}_{\alpha}$, $\chi^{\mathbf{t}}_{\beta}$ and $\chi^{\mathbf{f}}_{\alpha\wedge\beta}$.

Thus, we require that each entry in a truth table be consistent in the following sense.

Definition 1. *Let τ_1, τ_2, and τ be truth values in $\mathbf{T}_{6\text{v}}$, and let ω be a binary connective. We say that τ is* consistent *with ω on τ_1 and τ_2 if there exist propositional formulae α and β such that $\chi^{\tau_1}_{\alpha} \wedge \chi^{\tau_2}_{\beta} \wedge \chi^{\tau}_{\omega(\alpha,\beta)}$ is satisfiable. Similarly, τ is consistent with $\neg$ on τ_1 if there exists a propositional formula α such that $\chi^{\tau_1}_{\alpha} \wedge \chi^{\tau}_{\neg\alpha}$ is satisfiable.*

The notion of consistency directly yields the truth table of $\neg$ shown in Figure 3c, due to the following:

Proposition 1. *For every $\tau \in \mathbf{T}_{6\text{v}}$ there exists one and only one truth value in $\mathbf{T}_{6\text{v}}$ that is consistent with $\neg$ on τ.*

However, this is not the case for binary connectives: there are combinations of truth values that admit more than one consistent truth value, so consistency alone does not suffice to univocally define the truth tables for $\wedge$ and $\vee$. For example, both **f** and **sf** are consistent with $\mathbf{sf}\wedge\mathbf{sf}$, and both **t** and **st** are consistent with $\mathbf{st}\vee\mathbf{st}$. In such cases, how do we choose a suitable truth value? This is what we answer next.

Generality When there is more than one truth value that is consistent with a binary connective, we should pick the *most general* among them. To illustrate this point, let us consider the case of $\mathbf{sf}\wedge\mathbf{sf}$, which admits two consistent truth values: **sf** and **f**. Choosing **f** would preclude the existence of interpretations where the formula is true in some world. On the other hand, **sf** allows for this possibility without precluding the existence of interpretations where the formula is false in all worlds. We will make this intuition more precise in what follows.

For propositional interpretations $\mathcal{I} = (t, f, W)$ and $\mathcal{I}' = (t', f', W')$, we say that $\mathcal{I}$ is *more general* than $\mathcal{I}'$ (and write $\mathcal{I}' \preceq \mathcal{I}$), if there exists a surjective mapping $h\colon W \to W'$ such that, for every propositional formula α and every $w \in W$, all of the following hold:

- $w \in t(\alpha)$ implies $h(w) \in t'(\alpha)$, and
- $w \in f(\alpha)$ implies $h(w) \in f'(\alpha)$.

Intuitively, $\mathcal{I}$ is more general than $\mathcal{I}'$ if, for every propositional formula α, it has more worlds where α is not known to be true or false – that is, worlds that do not belong to either $t(\alpha)$ nor $f(\alpha)$ – but $\mathcal{I}$ agrees with $\mathcal{I}'$ on all the worlds for which this information is present.

Using this notion, we can define a partial ordering on subjective epistemic formulae as follows: we say that φ is *more general* than ψ (and write $\psi \preceq \varphi$) if for every model $\mathcal{I}$ of ψ there exists a model $\mathcal{I}'$ of φ such that $\mathcal{I} \preceq \mathcal{I}'$.

Finally, we can use generality to define a preference criterion for choosing a truth value over another when more than one are consistent with a connective.

Definition 2. *Let τ and τ' be truth values that are consistent with ω on τ_1 and τ_2. Then, τ' is* preferable *to τ with respect to $\omega(\tau_1, \tau_2)$ if*

$$\chi^{\tau_1}_{\alpha} \wedge \chi^{\tau_2}_{\beta} \wedge \chi^{\tau}_{\omega(\alpha,\beta)} \;\preceq\; \chi^{\tau_1}_{\alpha} \wedge \chi^{\tau_2}_{\beta} \wedge \chi^{\tau'}_{\omega(\alpha,\beta)}$$

for all propositional formulae α and β.

Of course, the above still leaves open the possibility that, among the truth values that are consistent with a binary connective, there might not be one that is preferable to all others. Below, we show that this is not the case.

Theorem 2. *Let $\omega \in \{\wedge, \vee\}$, let $\tau_1, \tau_2 \in \mathbf{T}_{6\text{v}}$, and let $\mathbf{C}$ be the subset of truth values in $\mathbf{T}_{6\text{v}}$ that are consistent with ω on τ_1 and τ_2. Then, there exists a unique $\tau \in \mathbf{C}$ such that, for every $\tau' \in \mathbf{C}$, τ is preferable to τ' with respect to $\omega(\tau_1, \tau_2)$.*

Thus, to define the truth table of a binary connective ω, for each combination of truth values τ_1 and τ_2 in $\mathbf{T}_{6\text{v}}$ we assign to $\omega(\tau_1, \tau_2)$ the most preferable truth value that is consistent with ω on τ_1 and τ_2. This yields the truth tables for $\wedge$ and $\vee$ shown in Figure 3a and 3b, respectively. Finally, we call $\mathbb{L}_{6\text{v}}$ the propositional logic consisting of the truth values in $\mathbf{T}_{6\text{v}}$ and the truth tables in Figure 3.

Coming back to the example of $\mathbf{sf}\wedge\mathbf{sf}$ mentioned earlier, we now illustrate intuitively why the requirement of generality is indeed reasonable. Suppose that two non-equivalent propositional formulae α and β are both assigned the truth value **sf**. If the evaluation is correct, then for every propositional interpretation there exists a world in which α is false and a world (not necessarily the same) in which β is false. Both **sf** and **f** are consistent with $\mathbf{sf}\wedge\mathbf{sf}$, so what truth value should $\alpha\wedge\beta$ evaluate to? The truth value **f** would indicate that $\alpha\wedge\beta$ is false in all worlds of every interpretation for which both α and β result in **sf**. Clearly, there are interpretations for which this happens, for example $(t, f, \{w_1, w_2\})$ with $f(\alpha) = \{w_1\}$, $f(\beta) = \{w_2\}$ and $t(\alpha) = t(\beta) = \varnothing$. However, there are also interpretations where this is not the case, for instance $(t', f', \{w_1, w_2)$ with $t'(\alpha) = t'(\beta) = \varnothing$ and $f'(\alpha) = f'(\beta) = \{w_1\}$. The truth value **sf** is general enough to correctly capture the outcome of $\mathbf{sf}\wedge\mathbf{sf}$ in all situations, including those mentioned above, while **f** may be incorrect in some cases.

SQL's Propositional Logic

The propositional logic $\mathbb{L}_{6\text{v}} = \big(\mathbf{T}_{6\text{v}}, \{\wedge, \vee, \neg\}\big)$ can express many nuances of the truth value of a propositional formula in the case of incomplete information. But can this logic be used in practice?

The query optimization engines of modern relational database management systems are based on decades of research that relies on a well established set of assumptions on the logic underlying the evaluation. Among these assumptions, there are two crucial properties of the binary connectives: *idempotency* and *distributivity*, see (Jarke and Koch 1984; Graefe 1993). These are used to transform redundant expressions into equivalent non-redundant ones, in order to reduce

$\wedge$	t	f	s	st	sf	u
t	t	f	s	st	sf	u
f	f	f	f	f	f	f
s	s	f	sf	sf	sf	sf
st	st	f	sf	u	sf	u
sf	sf	f	sf	sf	sf	sf
u	u	f	sf	u	sf	u

(a)

$\vee$	t	f	s	st	sf	u
t	t	t	t	t	t	t
f	t	f	s	st	sf	u
s	t	s	st	st	st	st
st	t	st	st	st	st	st
sf	t	sf	st	st	u	u
u	t	u	st	st	u	u

(b)

	$\neg$
t	f
f	t
s	s
st	sf
sf	st
u	u

(c)

Figure 3: The truth tables of $\mathbb{L}_{6v}$ for $\wedge$, $\vee$ and $\neg$.

the number of superfluous operations to be executed during query evaluation.

The binary connectives in $\mathbb{L}_{6v}$ are *weakly idempotent*, i.e., for every truth value $\tau \in \mathbf{T}_{6v}$ we have $\tau \wedge \tau \wedge \tau = \tau \wedge \tau$, and likewise for $\vee$. However, they are not idempotent: $\mathbf{s} \wedge \mathbf{s}$ and $\mathbf{s} \vee \mathbf{s}$ give **sf** and **st**, respectively, rather than **s**. Moreover, $\wedge$ does not distribute over $\vee$:

$$\underbrace{\mathbf{s} \wedge \overbrace{(\mathbf{s} \vee \mathbf{s})}^{\mathbf{st}}}_{\mathbf{sf}} \neq \underbrace{\overbrace{(\mathbf{s} \wedge \mathbf{s})}^{\mathbf{sf}} \vee \overbrace{(\mathbf{s} \wedge \mathbf{s})}^{\mathbf{sf}}}_{\mathbf{u}}$$

and $\vee$ does not distribute over $\wedge$:

$$\underbrace{\mathbf{s} \vee \overbrace{(\mathbf{s} \wedge \mathbf{s})}^{\mathbf{sf}}}_{\mathbf{st}} \neq \underbrace{\overbrace{(\mathbf{s} \vee \mathbf{s})}^{\mathbf{st}} \wedge \overbrace{(\mathbf{s} \vee \mathbf{s})}^{\mathbf{st}}}_{\mathbf{u}}$$

Due to the lack of idempotency and distributivity, $\mathbb{L}_{6v}$ is unlikely to be implemented in real systems for query evaluation. However, we can look for *sublogics* of $\mathbb{L}_{6v}$ with the desired properties.

To this end, we say that $\mathbb{L} = (\mathbf{T}, \Omega)$ is a sublogic of $\mathbb{L}' = (\mathbf{T}', \Omega')$ if $\mathbf{T} \subseteq \mathbf{T}'$ and for every $\omega \in \Omega$ there exists $\omega' \in \Omega'$ such that $\omega(\bar{\tau}) = \omega'(\bar{\tau})$ for every tuple of truth values in $\mathbf{T}$. A sublogic $\mathbb{L}$ of $\mathbb{L}'$ is *maximal with respect to a property* P if it has P and there is no sublogic $\mathbb{L}''$ of $\mathbb{L}'$ with property P such that $\mathbb{L}$ is a sublogic of $\mathbb{L}''$.

For practical purposes, we want a sublogic of $\mathbb{L}_{6v}$ that has the truth value **t** and it is maximal with respect to distributivity and idempotency.

Theorem 3. *$\mathbb{L}_{3v}$ is the only sublogic of $\mathbb{L}_{6v}$ that includes the truth value* **t** *and that is maximal with respect to distributivity and idempotency of the binary connectives ($\wedge$ and $\vee$).*

Therefore, when it comes to balancing expressiveness and practicality, the much criticized three-valued logic used by SQL is in fact a good choice for dealing with incomplete information in relational databases, at least for the propositional case.

We next examine extensions of propositional logics such as $\mathbb{L}_{6v}$ and $\mathbb{L}_{3v}$ to predicate logics.

Predicate Logics

As already explained, the need to consider predicate logics of incomplete information arises most commonly in querying incomplete databases, where special values – commonly referred to as *nulls* – indicate incompleteness of some sort. When atomic formulae may involve nulls – e.g., comparing a null with another value, or checking whether a tuple with nulls belongs to a relation – the standard approach is not to follow the Boolean semantics of FO, but instead to look for a many-valued semantics that will properly lift a propositional logic to all of FO. Such a semantics is by no means unique; we shall see three common versions later in this section.

We now define incomplete relational databases (which are in fact two-sorted relational structures), and consider many-valued FO logics on them, based on particular propositional logic. While propagating truth values through connectives and quantifiers is completely standard, assigning them to atoms is not unique. We consider three commonly occurring ways:

- one uses the Boolean semantics (Bolc and Borowik 1992),
- one adopts the approach of SQL (Date and Darwen 1996),
- and yet another is based on tuple unification, to achieve query answers with certainty guarantees (Libkin 2016).

As our main result, we show that in the context of many-valued FO, the exact choice of semantics of atoms, or truth values, or propositional connectives, does not matter: whatever combination of these one chooses, the resulting logic can be encoded in Boolean FO.

Incomplete Relational Structures (Databases)

As is standard in the database field and many applications of incomplete information, elements of relational structures (or relational databases; these terms are used interchangeably) come from two disjoint sets. One is the set Const of *constants*, i.e., known values that are stored in databases. The other is the set Null of *nulls* that represent unknown values. We always assume that Const is countably infinite. For the set Null, some options exist, of which the most common are the following.

- Null too is a countably infinite set. This corresponds to the model of *marked* nulls used both in relational databases and their many applications, such data exchange (Arenas et al. 2014), data integration (Lenzerini 2002) and OBDA (Bienvenu and Ortiz 2015).
- Null is a singleton set containing one element denoted by $\mathtt{N}$. This is the approach of SQL and implementations of relational DBMSs, where there is just one single null value.

A relational *vocabulary* σ (which is usually called *schema* in the database context) is a set $\{R_1, \ldots, R_n, =\}$ consisting

of relation names $R_1, \ldots, R_n$, each with an associated arity, plus a binary relation symbol "$=$" for equality. A *structure* $\mathfrak{A}$ of this vocabulary is a tuple $\langle A, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}}, =^{\mathfrak{A}} \rangle$, where:

- A is a finite subset of $\mathsf{Const} \cup \mathsf{Null}$,
- $R_i^{\mathfrak{A}} \subseteq A^k$ for every $i \in \{1, \ldots, n\}$, and
- $=^{\mathfrak{A}}$ is the binary relation defined as $\{(a, a) \mid a \in A\}$.

Many-valued Predicate Logics

A many-valued predicate logic is based on a many-valued propositional logic $\mathbb{L}$ with a set $\mathbf{T}$ of truth values and Ω of propositional connectives. Such a logic is a pair $\langle \mathrm{FO}(\mathbb{L}), [\![\,]\!] \rangle$ of FO formulae based on the propositional logic (to be defined shortly) and the semantics $[\![\,]\!]$ of its formulae. We assume throughout that $\mathbb{L}$ has connectives $\vee, \wedge$ which are commutative and associative. This is necessary to define quantifiers. Other connectives are arbitrary. We assume that truth values **t** and **f** are always included in $\mathbf{T}$.

Syntax and semantics of $\mathrm{FO}(\mathbb{L})$ Given a propositional logic $\mathbb{L}$ with truth values $\mathbf{T}$ and connectives Ω, formulae of $\mathrm{FO}(\mathbb{L})$ are defined by the following rules.

- Atomic formulae:
  - if R is a k-ary vocabulary symbol, and $x_1, \ldots, x_k$ are variables, then $R(x_1, \ldots, x_k)$ is an atomic formula; we shall also write the more common $x_1 = x_2$ in place of $=(x_1, x_2)$;
  - $\mathsf{const}(x)$ and $\mathsf{null}(x)$ are atomic formulae.
- If $\omega \in \Omega$ is a k-ary connective, and $\varphi_1, \ldots, \varphi_k$ are formulae, then $\omega(\varphi_1, \ldots, \varphi_k)$ is a formula.
- If φ is a formula and x is a variable, then $\exists x\, \varphi$ and $\forall x\, \varphi$ are formulae.

The notion of free variables is defined in the usual way.

The semantics of a formula φ is given with respect to a structure $\mathfrak{A}$ with universe A and an assignment ν of values in A to free variables of φ (i.e., ν is a partial function that is defined on all free variables of φ and takes values in A). This semantics will be denoted by $[\![\varphi]\!]_{\mathfrak{A},\nu}$, and it is a value in $\mathbf{T}$. In other words, $[\![\,]\!]$ assigns a truth value in $\mathbf{T}$ to φ in a structure $\mathfrak{A}$ under assignment ν.

The semantics of atoms const and null is as follows:

$$[\![\mathsf{const}(x)]\!]_{\mathfrak{A},\nu} = \begin{cases} \mathbf{t} & \text{if } \nu(x) \in \mathsf{Const}, \\ \mathbf{f} & \text{if } \nu(x) \in \mathsf{Null}. \end{cases}$$

$$[\![\mathsf{null}(x)]\!]_{\mathfrak{A},\nu} = \begin{cases} \mathbf{t} & \text{if } \nu(x) \in \mathsf{Null}, \\ \mathbf{f} & \text{if } \nu(x) \in \mathsf{Const}. \end{cases}$$

For propositional connectives and quantifiers, the semantics is defined with the *standard lifting rules*:

$$[\![\omega(\varphi_1, \ldots, \varphi_k)]\!]_{\mathfrak{A},\nu} = \omega([\![\varphi_1]\!]_{\mathfrak{A},\nu}, \ldots, [\![\varphi_k]\!]_{\mathfrak{A},\nu}),$$

$$[\![\exists x\, \varphi]\!]_{\mathfrak{A},\nu} = \bigvee_{a \in A} [\![\varphi]\!]_{\mathfrak{A},\nu[a/x]},$$

$$[\![\forall x\, \varphi]\!]_{\mathfrak{A},\nu} = \bigwedge_{a \in A} [\![\varphi]\!]_{\mathfrak{A},\nu[a/x]},$$

where $\nu[a/x]$ is the same as ν except that it assigns a to x. The last two rules rely on the fact that $\vee$ and $\wedge$ are commutative and associative.

For atomic formulae $R(\bar{x})$, with $R \in \sigma$, there are several options, which we now consider, when the underlying logic is either $\mathbb{L}_{\text{bool}}$ or $\mathbb{L}_{\text{3v}}$.

Boolean semantics This is the standard two-valued FO semantics, with only **t** and **f** as truth values, and it is given by

$$[\![R(\bar{x})]\!]^{\text{bool}}_{\mathfrak{A},\nu} = \begin{cases} \mathbf{t} & \text{if } \nu(\bar{x}) \in R^{\mathfrak{A}}, \\ \mathbf{f} & \text{if } \nu(\bar{x}) \notin R^{\mathfrak{A}}, \end{cases}$$

for every R in the vocabulary σ (which, recall, includes $=$). It is then extended to all of FO with the above rules, resulting in the semantics $[\![\,]\!]^{\text{bool}}$ defined for all FO formulae. When $[\![\varphi]\!]^{\text{bool}}_{\mathfrak{A},\nu} = \mathbf{t}$ we also write the more customary $\mathfrak{A}, \nu \models \varphi$.

The logic BFO, or *Boolean* FO, is now formally defined as $\mathrm{FO}(\mathbb{L}_{\text{bool}})$ interpreted under $[\![\,]\!]^{\text{bool}}$; it is the standard FO with only **t** and **f** as truth values.

Null-free semantics A tuple $\bar{a}$ is *null-free* if all of its values are from Const. The null-free semantics of $\mathrm{FO}(\mathbb{L}_{\text{3v}})$ is the same as the Boolean semantics for tuples of constants; if any nulls are present, it produces the truth value **u**:

$$[\![R(\bar{x})]\!]^{\text{nf}}_{\mathfrak{A},\nu} = \begin{cases} \mathbf{t} & \text{if } \nu(\bar{x}) \in R^{\mathfrak{A}} \text{ and } \nu(\bar{x}) \text{ is null-free}, \\ \mathbf{f} & \text{if } \nu(\bar{x}) \notin R^{\mathfrak{A}} \text{ and } \nu(\bar{x}) \text{ is null-free}, \\ \mathbf{u} & \text{if } \nu(\bar{x}) \text{ contains a null}, \end{cases}$$

for every R in the vocabulary σ (which, recall, includes $=$). In particular, for the equality predicate $=$, this is exactly the semantics used by SQL (Date and Darwen 1996).

Unification semantics A semantics based on the notion of tuple unification was proposed by Libkin (2016) to enforce certainty guarantees for query answers. We say that two tuples $\bar{a}$ and $\bar{b}$ *unify* if there is a map $h\colon \mathsf{Const} \cup \mathsf{Null} \to \mathsf{Const}$ that is the identity on constants and such that $h(\bar{a}) = h(\bar{b})$. Then, for every relation symbol R in the vocabulary σ, the unification semantics is defined by

$$[\![R(\bar{x})]\!]^{\Uparrow}_{\mathfrak{A},\nu} = \begin{cases} \mathbf{t} & \text{if } \nu(\bar{x}) \in R^{\mathfrak{A}}, \\ \mathbf{f} & \text{if } \nexists \bar{a} \in R^{\mathfrak{A}} \text{ s.t. } \nu(\bar{x}) \text{ and } \bar{a} \text{ unify}, \\ \mathbf{u} & \text{otherwise}. \end{cases}$$

The semantics $[\![\,]\!]^{\Uparrow}$ is then lifted to all of FO by the standard lifting rules.

The reason this semantics was introduced is that it ensures certainty of answers to FO queries: if $[\![\varphi(\bar{x})]\!]^{\Uparrow}_{\mathfrak{A},\nu} = \mathbf{t}$, then the tuple $\bar{u} = \nu(\bar{x})$ is what is known as a certain answer to φ, i.e., $h(\mathfrak{A}) \models \varphi\big(h(\bar{u})\big)$ for every map $h : \mathsf{Const} \cup \mathsf{Null} \to \mathsf{Const}$ that is the identity on constants.

Mixed semantics There is a priori no reason to apply the same semantics on each relation symbol $R \in \sigma$; instead we can freely mix them. A mixed semantics $[\![\,]\!]^s$ is then given by a function $s : \sigma \to \{\text{bool}, \Uparrow, \text{nf}\}$ so that $[\![R(\bar{x})]\!]^s_{\mathfrak{A},\nu} = [\![R(\bar{x})]\!]^{s(R)}_{\mathfrak{A},\nu}$. This generalizes Boolean, unification, and null-free semantics.

Boolean FO Captures Many-valued FO

We now show that in most cases, many-valued predicate logics do not give any extra power compared to BFO, i.e., the usual FO under the standard Boolean interpretation of connectives and the Boolean semantics of atomic formulae. The notion of capturing a many-valued FO logic by BFO is defined as follows.

Definition 3. *A formula φ of* $\mathrm{FO}(\mathbb{L})$ *over a many-valued propositional logic $\mathbb{L}$ with truth values* $\mathbf{T}$ *is* captured *by* BFO *under semantics $[\![\,]\!]$ if there exist* BFO *formulae φ_τ for each $\tau \in \mathbf{T}$ such that for every structure $\mathfrak{A}$ and assignment ν of free variables of φ we have*

$$[\![\varphi]\!]_{\mathfrak{A},\nu} = \tau \;\Leftrightarrow\; \mathfrak{A}, \nu \models \varphi_\tau\,.$$

$\mathrm{FO}(\mathbb{L})$ *is captured by* BFO *if each of its formulae is.*

Usually we are interested in formulae that are true in a given structure, i.e., $[\![\varphi]\!]_{\mathfrak{A},\nu} = \mathbf{t}$. If a formula is captured by BFO, this tells us that we do not need many-valued semantics, and instead can simply check whether $\mathfrak{A}, \nu \models \varphi_{\mathbf{t}}$ under the usual Boolean semantics.

To capture a many-valued FO by BFO we need very few assumptions. Recall that $\mathbb{L} = \langle \mathbf{T}, \Omega \rangle$ is given by a set of truth values and truth tables for connectives in Ω, which we assume to contain at least $\vee, \wedge$ to define quantifiers. In logics such as $\mathbb{L}_{\text{bool}}$ and $\mathbb{L}_{3\text{v}}$, these connectives are *idempotent*, i.e., $\tau \wedge \tau = \tau \vee \tau = \tau$ for every $\tau \in \mathbf{T}$. In $\mathbb{L}_{6\text{v}}$, they are *weakly idempotent*: $\tau \wedge \tau \wedge \tau = \tau \wedge \tau$ and likewise for $\vee$. Notice that idempotency implies weak idempotency. This is the only condition we need to impose to be able to lift capturing formulae by Boolean FO from atoms to arbitrary formulae.

Theorem 4. *Let $\mathbb{L}$ be a propositional many-valued logic in which connectives $\wedge$ and $\vee$ are weakly idempotent. Assume that every relational atom $R(\bar{x})$, for $R \in \sigma$, is captured by* BFO *under $[\![\,]\!]$. Then every* $\mathrm{FO}(\mathbb{L})$ *formula over vocabulary σ is captured by* BFO *under $[\![\,]\!]$.*

To apply this result to the previously considered semantics, we need to capture atomic formulae, under different semantics, in BFO. This is possible for all of them.

Proposition 2. *Relational atoms are captured by* BFO *under Boolean, unification, and null-free semantics.*

Finally, this tells us that any mixed semantics (including its pure versions, i.e., Boolean, unification, null-free) coupled with any propositional many-valued logic like $\mathbb{L}_{3\text{v}}$ or $\mathbb{L}_{6\text{v}}$ (as long as it has weakly idempotent conjunction and disjunction) is no more powerful than the standard semantics over two truth values $\mathbf{t}$ and $\mathbf{f}$.

Corollary 1. *Let $\mathbb{L}$ be a propositional many-valued logic whose truth values include $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, with an arbitrary set of connectives where $\vee$ and $\wedge$ are (weakly) idempotent. Then for every vocabulary σ, every function s defining a mixed semantics, and every formula φ of* $\mathrm{FO}(\mathbb{L})$ *there is a formula φ' of* BFO *such that $[\![\varphi]\!]^s_{\mathfrak{A},\nu} = \mathbf{t}$ iff $\mathfrak{A}, \nu \models \varphi'$.*

Using this result, we can clarify, in the next section, the question of the power of the logic that underlies real-life database applications that use incomplete information.

The Logic of SQL

Most database texts will claim that the core of SQL, the main relational database query language, is first-order logic FO. This was certainly true in the early stages of SQL design, as it grew out of relational calculus, which is just another name for FO. But then the language gained many features, in particular null values, leading to more complex underlying logics.

These logics are still not well understood, as the formalization of SQL mainly took a different route via *relational algebra*, which is the procedural counterpart of FO. Several attempts to provide a theoretical language behind SQL looked at relational algebra translations of the language (Ceri and Gottlob 1985; Van den Bussche and Vansummeren 2009) or presented semantics of various fragments of the language, often under the simplifying assumption that no nulls are present and no three-valued logic is used (Chu et al. 2017; Negri, Pelagatti, and Sbattella 1991). An attempt to find a logic underlying SQL concentrated on its features that go beyond FO (i.e., aggregation) rather than nulls (Hella et al. 2001). More recent work (Guagliardo and Libkin 2017), while providing a direct semantics of SQL, accounted for null values and three-valued logic, and even gave a translation of SQL queries that, similarly in spirit to the results in the previous section, showed how to evaluate them without ever producing the unknown truth value $\mathbf{u}$. This was done, however, at the level of SQL queries. We now analyze the power of SQL and the need for three truth values at a purely logical level.

We start with the basic fragment of relational languages that has the power of FO, or – equivalently – the basic operations of relational algebra, or SQL's select-from-where queries without aggregation. These operate on databases whose values come from Const. Recall that SQL uses a single null denoted here by $\mathbf{N}$. Now we add it; how should the logic change to capture this extension? It depends on who is asked to produce such an extension.

A logician's approach If the domain is extended by a single constant, we simply consider FO over $\mathsf{Const} \cup \{\mathbf{N}\}$ with a unary predicate $\mathsf{null}(\,)$ that is only true in $\mathbf{N}$ (to keep the vocabulary relational; alternatively a constant symbol could be added). The interpretation of $=$ is simply $\{(c,c) \mid c \in \mathsf{Const}\} \cup \{(\mathbf{N},\mathbf{N})\}$, i.e., *syntactic equality*: $\mathbf{N}$ is equal to itself, and not equal to any element of Const. In other words, the logic is the usual BFO, with all the atoms interpreted under the Boolean semantics $[\![\,]\!]^{\text{bool}}$.

It would thus be seen, by a logician, as an overkill to introduce a many-valued logic to deal with just one extra element of the domain. Nonetheless, this is what SQL did.

SQL approach: a textbook version The usual explanation of the logic behind SQL is that it adds a new truth value $\mathbf{u}$ to account for any comparisons involving nulls. In other words, the logic is $\mathrm{FO}(\mathbb{L}_{3\text{v}})$, and the semantics $[\![\,]\!]^{\text{sql}}$ is mixed, combining Boolean and null-free semantics:

- for relational atoms, $[\![R(\bar{x})]\!]^{\text{sql}}_{\mathfrak{A},\nu} = [\![R(\bar{x})]\!]^{\text{bool}}_{\mathfrak{A},\nu}$;

- for equality, $[\![x = y]\!]^{\text{sql}}_{\mathfrak{A},\nu} = [\![x = y]\!]^{\text{nf}}_{\mathfrak{A},\nu}$.

SQL approach: what really happens While the textbook approach comes close to describing the logic of SQL, it misses one important feature of such logic. In essence, we can think of SQL queries as expressions

$$\begin{array}{ll}\mathsf{select} & \bar{x} \\ \mathsf{from} & Q_1, \ldots, Q_n \\ \mathsf{where} & \theta(\bar{x}_1, \ldots, \bar{x}_n)\end{array}$$

where $Q_1, \ldots, Q_n$ are either queries or relations, $\bar{x}_i$ is a tuple of variables returned by Q_i, and θ is a condition composed of equalities of variables and constants, or statements $Q'(\bar{y})$, where Q' is another query, or statements $Q' \neq \varnothing$, combined using $\wedge$, $\vee$, and $\neg$.

Note that in SQL query evaluation, it is the conditions θ that are evaluated in $\mathbb{L}_{3\text{v}}$; once the evaluation of the $\mathsf{where}\ \theta$ clause is finished, only tuples that evaluated to **t** are kept. To capture this in logic, we need a propositional operator that collapses **f** and **u** into **f**. Such an operator does exist in propositional many-valued logics (Bochvar 1981) and is known as an *assertion* operator: $\uparrow p$ for a proposition p evaluates to **t** if p evaluates to **t**, and to **f** otherwise. Let $\mathbb{L}^{\uparrow}_{3\text{v}}$ be the extension of $\mathbb{L}_{3\text{v}}$ with this operator.

The basic SQL query can then be expressed in $\text{FO}(\mathbb{L}^{\uparrow}_{3\text{v}})$:

$$Q(\bar{x}) \;=\; \exists \bar{y} \bigwedge_{i=1}^{n} Q_i(\bar{x}_i) \wedge \uparrow\theta(\bar{x}_1, \ldots, \bar{x}_n)\,,$$

where $\bar{y}$ lists the variables present in $\bar{x}_1, \ldots, \bar{x}_n$ but not in $\bar{x}$. Thus, the many-valued predicate logic capturing SQL's behavior is $\text{FO}(\mathbb{L}^{\uparrow}_{3\text{v}})$ under $[\![\,]\!]^{\text{sql}}$.

To sum up, there are three choices of a logic capturing SQL's behavior:

1) Boolean predicate logic BFO;
2) FO based on Kleene's logic under the $[\![\,]\!]^{\text{sql}}$ semantics;
3) FO based on Kleene's logic with the assertion operator under the $[\![\,]\!]^{\text{sql}}$ semantics.

These logics use different sets of truth values. However, it only matters when formulae evaluate to true, as this determines the output of queries. Thus, to compare logics with different sets of truth values, we say that two logics, $\text{FO}(\mathbb{L}_1)$ under $[\![\,]\!]^1$, and $\text{FO}(\mathbb{L}_2)$ under $[\![\,]\!]^2$, are *true-equivalent* if the models of **t** are the same in both. That is, for every formula φ_1 of $\text{FO}(\mathbb{L}_1)$ there is a formula φ_2 of $\text{FO}(\mathbb{L}_2)$ such that

$$[\![\varphi_1]\!]^1_{\mathfrak{A},\nu} = \mathbf{t} \;\;\Leftrightarrow\;\; [\![\varphi_2]\!]^2_{\mathfrak{A},\nu} = \mathbf{t}$$

for every $\mathfrak{A}, \nu$, and vice versa, for each φ_2 of $\text{FO}(\mathbb{L}_2)$ there is a formula φ_1 of $\text{FO}(\mathbb{L}_1)$ such that the above condition holds.

Then, with respect to the truth value **t**, there is no difference between the logics that attempt to model SQL's behavior.

Theorem 5. *The logics* $\text{FO}(\mathbb{L}_{3\text{v}})$ *and* $\text{FO}(\mathbb{L}^{\uparrow}_{3\text{v}})$*, both under* $[\![\,]\!]^{\text{sql}}$*, and* BFO*, are all true-equivalent.*

Thus, the most natural logical approach to adding a null value to the language does not miss any expressiveness of the more complex solutions based on many-valued logics.

Conclusions

To conclude, let us revisit history. Handling incomplete information by logical languages is an important topic, especially in data management. All commercial database systems that speak SQL offer a solution based on a three-valued propositional logic that is lifted then to full predicate logic. This solution was heavily criticized in the literature, but at the level of the chosen propositional logic.

We proposed a principled approach to justifying a proper logic for handling incomplete information, which resulted in a six-valued logic $\mathbb{L}_{6\text{v}}$. However, taking into account the needs of SQL query evaluation (e.g., distributivity laws), the largest fragment of $\mathbb{L}_{6\text{v}}$ that does not break traditional evaluation and optimization strategies is Kleene's logic $\mathbb{L}_{3\text{v}}$, precisely the one chosen by SQL.

However, even though the SQL designers were justified in their choice of Kleene's logic, they neglected to consider the impact that lifting it to full predicate logic would have. We showed that it leads to no increase in expressive power; had this been known to the SQL designers, perhaps other choices would have been considered too.

But does this mean that we should abandon many-valued logics of incomplete information? Most likely not: while the theoretical complexity of formulae that result from eliminating many-valuedness is the same as that of original many-valued formulae, their *practical* complexity (i.e., if implemented as real life database queries) is likely to be different. This is mainly due to the fact that 40 years of research on query evaluation and optimization had one particular model in mind, and that model used a many-valued logic. However, the observations we made here might have an impact on the design of new languages, since avoiding many-valued logics for handling incompleteness is now an option.

Regarding future directions, we would like to extend the propositional setup with bilattice orderings as is often done (Arieli and Avron 1996; Ginsberg 1988), and understand the right orderings for logics like $\mathbb{L}_{6\text{v}}$. Yet another direction is to drop the restriction $t(\alpha) \cap f(\alpha) = \varnothing$ for every propositional formula α. Such restrictions have been lifted in the study of paraconsistent logics (Arieli, Avron, and Zamansky 2010; Zamansky and Avron 2006), and in fact the question of looking for the right many-valued logic for reasoning about inconsistency has been raised (Arieli, Avron, and Zamansky 2011). Our focus would be slightly different, as we want to extend the current study to handle the most common case of inconsistency in data management, namely inconsistency with respect to integrity constraints (Arenas, Bertossi, and Chomicki 1999; Bertossi 2011).

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments. This work was partly supported by EPSRC grants M025268 and N023056.

References

Arenas, M.; Bertossi, L.; and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *ACM Symposium on Principles of Database Systems (PODS)*, 68–79.

Arenas, M.; Barceló, P.; Libkin, L.; and Murlak, F. 2014. *Foundations of Data Exchange*. Cambridge University Press.

Arieli, O., and Avron, A. 1996. Reasoning with logical bilattices. *Journal of Logic, Language and Information* 5(1):25–63.

Arieli, O., and Avron, A. 1998. The logical role of the four-valued bilattice. In *LICS*, 118–126.

Arieli, O.; Avron, A.; and Zamansky, A. 2010. Maximally paraconsistent three-valued logics. In *KR*.

Arieli, O.; Avron, A.; and Zamansky, A. 2011. What is an ideal logic for reasoning with inconsistency? In *IJCAI*, 706–711.

Belnap, N. D. 1977. A useful four-valued logic. In Dunn, J. M., and Epstein, G., eds., *Modern Uses of Multiple-Valued Logic*. D. Reidel. 8–37.

Bertossi, L. 2011. *Database Repairing and Consistent Query Answering*. Morgan&Claypool Publishers.

Bienvenu, M., and Ortiz, M. 2015. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web*, 218–307.

Bochvar, D. A. 1981. On a three-valued logical calculus and its application to the analysis of the paradoxes of the classical extended functional calculus. *History and Philosophy of Logic* 2:87–112. Translated from *Matematicheskij Sbornik*, 4 (46): 287-308, 1938.

Bolc, L., and Borowik, P. 1992. *Many-Valued Logics: Theoretical Foundations*. Springer.

Ceri, S., and Gottlob, G. 1985. Translating SQL into relational algebra: Optimization, semantics, and equivalence of SQL queries. *IEEE Trans. Software Eng.* 11(4):324–345.

Chu, S.; Weitz, K.; Cheung, A.; and Suciu, D. 2017. HoTTSQL: Proving query rewrites with univalent SQL semantics. In *PLDI*, 510–524. ACM.

Codd, E. F. 1975. Understanding relations (installment #7). *FDT - Bulletin of ACM SIGMOD* 7(3):23–28.

Codd, E. F. 1987. More commentary on missing information in relational databases. *SIGMOD Record* 16(1):42–50.

Console, M.; Guagliardo, P.; and Libkin, L. 2016. Approximations and refinements of certain answers via many-valued logics. In *KR*, 349–358.

Darwen, H., and Date, C. J. 1995. The third manifesto. *SIGMOD Record* 24(1):39–49.

Date, C. J., and Darwen, H. 1996. *A Guide to the SQL Standard*. Addison-Wesley.

Date, C. J. 2005. *Database in Depth - Relational Theory for Practitioners*. O'Reilly.

Fitting, M. 1991. Kleene's logic, generalized. *J. Log. Comput.* 1(6):797–810.

Gessert, G. H. 1990. Four valued logic for relational database systems. *SIGMOD Record* 19(1):29–35.

Ginsberg, M. L. 1988. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence* 4:265–316.

Graefe, G. 1993. Query evaluation techniques for large databases. *ACM Comput. Surv.* 25(2):73–170.

Grahne, G.; Moallemi, A.; and Onet, A. 2015. Intuitionistic data exchange. In *9th Alberto Mendelzon International Workshop on Foundations of Data Management*.

Guagliardo, P., and Libkin, L. 2017. A formal semantics of SQL queries, its validation, and applications. *PVLDB* 11(1):27–39.

Hella, L.; Libkin, L.; Nurmonen, J.; and Wong, L. 2001. Logics with aggregate operators. *Journal of the ACM* 48(4):880–907.

ISO/IEC. 2016. *ISO/IEC 9075:2016: Information technology – Database languages – SQL*. International Organization for Standardization.

Jarke, M., and Koch, J. 1984. Query optimization in database systems. *ACM Comput. Surv.* 16(2):111–152.

Lenzerini, M. 2002. Data integration: a theoretical perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, 233–246.

Libkin, L. 2016. SQL's three-valued logic and certain answers. *ACM Trans. Database Syst.* 41(1):1:1–1:28.

Negri, M.; Pelagatti, G.; and Sbattella, L. 1991. Formal semantics of SQL queries. *ACM Trans. Database Syst.* 16(3):513–534.

Reiter, R. 1980. A logic for default reasoning. *Artif. Intell.* 13(1-2):81–132.

Van den Bussche, J., and Vansummeren, S. 2009. Translating SQL into the relational algebra. Course notes, Hasselt University and Université Libre de Bruxelles.

Yue, K. 1991. A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Record* 20(3):43–49.

Zamansky, A., and Avron, A. 2006. Non-deterministic semantics for first-order paraconsistent logics. In *KR*, 431–439.